Design Rationale

Object-Oriented Programming Principle implementations:

New Class Implementations:

S O U L S

Souls class inherits from the Item class as Items represent an object in the Gameworld. As it is picked up and dropped which can be done through utilising getPickUpAction() and getDropAction() method in Items. When Souls are dropped by dead Enemies, it is instantiated, allowing the player to perform the method getPickUpAction(). This is done to prevent repetitive code. Hence, the Don't Repeat Yourself (DRY) principle, where a piece of logic will have its unique and specific functions within a system. There will also be a TokenOfSouls attribute that stores the Souls for when the player dies, meaning the number of Souls in the inventory of the Player before they die is stored in this attribute and removed from Player's inventory using removeItemFromInventory() inherited from Actor class.

E N E M Y

Enemy class is an abstract class extending the Actor class, representing the children classes and acts as a base code for its child classes: Undead, Skeletons and Lord of Cinder. To ensure that an instance of the Enemy class is not instantiated. The death of Enemies will drop Souls, the currency of the game. Each Enemy will have the base Souls count of 50 for Undead, 250 for Skeletons and 5000 for Lord of Cinder. These values are stored as a constructor parameter for creating instances of each enemy. The main objective is for the Player to pick up the Souls that are dropped by Enemies after they are killed and added into the Player's inventory and most importantly, the attacks between the Player and Enemy. The damages dealt by the Enemy can be stored in this class to prevent duplicate code in the Enemy children classes, instead, the children classes can just override this method. This is again an example of the implementation of the DRY principle as well as avoiding the excessive use of literals as it is not hard-coded, making the maintenance of the code smoother and easier in the future.

Enemy class is created to follow the Reduce Dependency Principle as the Actor class is initially inherited by the Player, Undead, Skeleton, LordOfCinder and Ground class. Hence, the Enemy class will inherit the Actor class instead, which is then inherited by its children classes like the Undead and EnemyWithWeapon (inherited by Skeletons and Yhrom) reducing dependency on the Actor class.

Enemy class will also be inclusive of an Arraylist of the Actor data type, keeping track of if the Actor is a Player or an Enemy is close together to determine the behaviour. This attribute will be looped through to determine if Enemy will use AttackAction() if the Player is in the target range. Thus, also reducing the use of duplicate code and utilising the DRY principle.

S K E L E T O N

Skeleton class will extend the EnemyWithWeapon class as it deals damage and displays phrases like "thwacks" or "punches". Each skeleton will also deal base damage to the player signifying how it will override the method indirectly inherited from the Enemy class. This class will implement a spawnEnemies() method which spawns 4 to 12 skeletons on the map. It is given the range of Skeletons that need to be spawned on the map and utilises WanderBahvaiour() to place the Skeleton in a random location of the map. As well as using the revived() method Skeletons class will inherit the parent constructor and will have these values as parameters: 100 hitpoints, 250Souls and a Weapon.

E N E M Y W I T H W E A P O N

EnemyWithWeapon class is implemented and inherits the Enemy class because the Skeletons and Yhorm wield Weapons, unlike the Undead. This implementation will follow the Liskov Substitution Principle (LSP) as the constructor in the Enemy Class, will have a parameter asking for weapons that the Enemy is possibly wielding. This demonstrates the superclass object being replaced by the subclass object without affecting the functionality of the superclass. Like how implementing the EnenyWithWeapon class will not interfere with the ability of the Enemy to attack the Player.

B O N F I R E

Bonfire is an extension of Ground. Overriding the canActorEnter() method and setting the return boolean value to False if the Actor is an Enemy. It is also responsible for recharging the Player' number of Estus Flask if it is less than 3. A new method rechargeEstusFlask() can be done in this class, where the boolean method checks if the Player has less than 3 Estus Flasks, if it returns True, then it is recharged to 3 Estus Flaks; else, do nothing and return a boolean value of False. This class is also responsible for resetting the Enemy positions once the Player dies. The vendor extends this class as the Vendor's actions are done on this terrain, Bonfire.

V E N D O R

This class will hold the methods in order for the Player to trade Souls for upgrades or Weapons. Vendor implements Souls as Souls is the object that interacts between Player and Vendor for the upgrades or weapon purchases. As the Player purchases a Weapon, the SwapWeaponAction which is a part of PickUpItemAction() is then called. This occurs as the Player can only wield one weapon at a time.

C E M E T E R Y

Cemetery class extends Ground class as the class represents the different Ground-type. The cemetery has a special trait where the Undead are spawned in this terrain type. A requirement for this class would also be that there will be at least 5 Cemeteries in the GameMap. Having this being a requirement, a method implemented is required to check the number of Cemeteries within the GameMap, if it is less than 5 the newGround() method in FancyGroundFectory class will be called to create more Cemetery Grounds. This is again an example of the DRY principle where codes is not repeated as the newGround() is called instead of making a new method specifically for creating new Grounds in the GameMap.

New Method Implementations:

TargetActorInRange()

With the return type of boolean. This method is implemented to check if the Player is within range of the Enemy, when the boolean return True, the Enemy will use AttackAction() to attack the Player (Unkindled), decreasing hitPoints. Target Actor refers to Player and this method is implemented in the Enemy class as all types of Enemy will check if the Player is within its range.

SpawnEnemies()

As mentioned above, this method is responsible for spawning the targeted number of enemies in the GameMap. Nested under the Enemy class, it is given the number range of Enemy type that needs to be spawned.

HasRevived()

This boolean return type method is implemented in the Skeleton class. Skeletons have a 50% success rate of healing themselves. When the method returns False, meaning the Skeleton has not revived itself, it will heal to its max_heal_points. Else, when the method returns True, the Skeleton is then removed from the GameMap.

HasBurn()

This method is created under the LordOfCinder class and checks for it Yhorm's hitpoints are less than 50%. So the method will first convert the hitPoints to a percentage which is then used to check if the hitpoints are less than 50%, If it is less than 50%, then Yhorm is able to perform an Action which he burns the Dirt terrain close him, affecting Player's hitpoints.